Language Quotients Revisited

Stavros Konstantinidis*
Saint Mary's University. 923, Robie Str, Halifax, Nova Scotia, Canada
s.konstantinidis@smu.ca

Nelma Moreira and Rogério Reis †
CMUP & DCC, Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 4169-007 Porto, Portugal
{*nelma.moreira,rogerio.reis*}*@fc.up.pt*

We revisit the basic concept of quotient of a regular language R by a language L that is not necessarily regular. We revise the deterministic complexity upper bounds of the quotient operation, and we also address the nondeterministic case. Specifically, the revised deterministic upper bound is shown to be more accurate for all hard streams of regular languages. When both languages R and L are regular, we give algorithms to construct their quotient in four ways. If the two languages are given via NFAs, the first algorithm produces an NFA for the desired quotient. If the two languages are given via regular expressions, we present an algorithm that produces a regular expression for the quotient both using ordinary derivatives and using partial derivatives. Finally, we consider regular expressions with a quotient operator and define the set of partial derivatives for regular expressions with this operator. Thus, using the partial derivative automaton, one can directly produce an NFA for the quotient. We have implemented all algorithms and present here experimental results.

Keywords: Regular languages; Regular expressions; Derivatives; Partial derivatives; Quotients of languages; Automata; State Complexity.

1. Introduction

Brzozowski [5] defined the *quotient complexity* of a language L as the number of distinct languages that are left-quotients of L by a word w, i.e., $w^{-1}L$. It is well known that a language is regular if and only if it has a finite number of left-quotients by a word. The left-quotient of a language L w.r.t to a language L' is defined as the union of the left-quotients of L by the words of L', i.e., ${L'}^{-1}L$. Again, if L is regular then ${L'}^{-1}L$ is regular for every language L'. In such case, the language quotients ${L'}^{-1}L$ were studied in 1960s by Conway [9, Chapter 5] by the name of *event derivatives*. Event derivatives extended derivatives of regular expression by a word introduced by Brzozowski [7], pre-bases studied by Mirkin [14] and, later, partial derivatives

*This work was partially supported by NSERC, Canada.
†This work was partially supported by CMUP, member of LASI, which is financed by national funds through FCT – Fundação para a Ciência e a Tecnologia, I.P., under the projects with reference UIDB/00144/2020 and UIDP/00144/2020.

defined by Antimirov [2]. The state complexity of language quotients was studied by Yu, Zhuang, and Salomaa [18]. For regular languages, state complexity coincides with quotient complexity. In [5] it was shown that quotient complexity can be used in a convenient manner to study upper bounds of state complexity. Gruber and Holzer studied some language operations related to quotients [11]. Derivatives and partial derivatives provide also tools to decide problems such as the word membership, universality, or equivalence of regular expressions, avoiding the automaton construction.

In this paper, we revisit the quotient operation between two languages. When a language R is regular, we revise the deterministic complexity upper bounds of the quotient operation $L^{-1}R$, and address the nondeterministic case. Specifically, the revised deterministic upper bound is shown to be more accurate for all hard streams of regular languages. If both languages are regular, we give algorithms to construct their quotient in four ways. If the two languages are given via NFAs, the first algorithm produces an NFA for the desired quotient. If the two languages are given via regular expressions, we present an algorithm that produces a regular expression for the quotient, both using ordinary derivatives and using partial derivatives. Finally, we consider regular expressions with a quotient operator and define the set of partial derivatives for regular expressions with this operator. Thus, one can directly produce an NFA for the quotient using the partial derivative automaton. Finally, we present some experimental results for the performance of the developed algorithms.

2. Preliminares

In this section, we review some basic definitions about languages, regular expressions, and automata. Given an alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ of size $k \geq 1$, a *language* L is a subset of the free monoid $\Sigma^\star$. We assume that Σ is ordered in the natural way, i.e., $\sigma_{i+1} \geq \sigma_i$ for $i = 1, \ldots, k-1$. The empty word is denoted by ε. The length of a word w is denoted by $|w|$. Given two words w_1, w_2, we say that w_1 is *lexicographically* less than w_2 if, either $|w_1| < |w_2|$, or $|w_1| = |w_2|$ and the words can be written as $w_1 = x\sigma_i y_1$, $w_2 = x\sigma_j y_2$ with $\sigma_i < \sigma_j$.

The set RE of (standard) *regular expressions* $\boldsymbol{r}$ over Σ is defined by the following context-free grammar:

$$\boldsymbol{r} := \emptyset \mid \varepsilon \mid \sigma_1 \mid \cdots \mid \sigma_k \mid (\boldsymbol{r} + \boldsymbol{r}) \mid (\boldsymbol{r} \cdot \boldsymbol{r}) \mid (\boldsymbol{r}^\star), \tag{1}$$

where the symbol $\cdot$ is normally omitted. The (regular) *language* represented by an expression $\boldsymbol{r} \in RE$ is denoted by $\mathcal{L}(\boldsymbol{r})$ and is inductively defined as follows for $\boldsymbol{r}, \boldsymbol{r}' \in RE$: $\mathcal{L}(\emptyset) = \emptyset$, $\mathcal{L}(\varepsilon) = \{\varepsilon\}$, $\mathcal{L}(\sigma) = \{\sigma\}$, $\mathcal{L}(\boldsymbol{r} + \boldsymbol{r}') = \mathcal{L}(\boldsymbol{r}) \cup \mathcal{L}(\boldsymbol{r}')$, $\mathcal{L}(\boldsymbol{r}\boldsymbol{r}') = \mathcal{L}(\boldsymbol{r})\mathcal{L}(\boldsymbol{r}') = \{\, wv \mid w \in \mathcal{L}(\boldsymbol{r}) \wedge v \in \mathcal{L}(\boldsymbol{r}') \,\}$, and $\mathcal{L}(\boldsymbol{r}^\star) = \mathcal{L}(\boldsymbol{r})^\star = \bigcup_{n \in \mathbb{N}} (\mathcal{L}(\boldsymbol{r})^n)$. For the *size* of a regular expression $\boldsymbol{r}$, denoted by $\|\boldsymbol{r}\|$, we consider the size of its syntactic tree, i.e., the number of symbols in $\boldsymbol{r}$, not counting parentheses but including the concatenation. The *alphabetic size* of $\boldsymbol{r}$, denoted by $|\boldsymbol{r}|_\Sigma$, is just the number of alphabetic symbols in $\boldsymbol{r}$. We define $\varepsilon(\boldsymbol{r}) = \varepsilon$ if $\varepsilon \in \mathcal{L}(\boldsymbol{r})$, and $\varepsilon(\boldsymbol{r}) = \emptyset$,

otherwise. In the first case, we say that $\boldsymbol{r}$ is *accepting* otherwise, it is *rejecting*. The function $\varepsilon()$ can be defined inductively on the structure of $\boldsymbol{r}$: $\varepsilon(\emptyset) = \varepsilon(\sigma) = \emptyset$; $\varepsilon(\varepsilon) = \varepsilon(\boldsymbol{r}^\star) = \varepsilon$; $\varepsilon(\boldsymbol{r} + \boldsymbol{r}') = \varepsilon$ if $\varepsilon(\boldsymbol{r}) = \varepsilon$ or $\varepsilon(\boldsymbol{r}') = \varepsilon$, being $\emptyset$ otherwise; and $\varepsilon(\boldsymbol{r}\boldsymbol{r}') = \varepsilon$ if $\varepsilon(\boldsymbol{r}) = \varepsilon(\boldsymbol{r}') = \varepsilon$, being $\emptyset$ otherwise. This function can be extended in a natural way to sets of expressions, languages, and other language operations. Two expressions $\boldsymbol{r}$ and $\boldsymbol{r}'$ are *equivalent* if their languages are the same, in which case we write $\boldsymbol{r} \doteq \boldsymbol{r}'$. With this interpretation, the algebraic structure $(RE, +, \cdot, \emptyset, \varepsilon)$ is an idempotent semiring that with $\star$ forms a Kleene algebra.

A *nondeterministic finite automaton* (NFA) is a quintuple $\boldsymbol{a} = \langle Q, \Sigma, \delta, I, F\rangle$ where Q is a finite set of states, Σ is a finite alphabet, $I \subseteq Q$ is the non-empty set of initial states, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Sigma \to 2^Q$ is the transition function. If $p \in \delta(q, \sigma)$ we say that (q, σ, p) is a *transition* of $\boldsymbol{a}$. The *language accepted* by $\boldsymbol{a}$ is $\mathcal{L}(\boldsymbol{a}) = \{\, w \in \Sigma^\star \mid \delta(I, w) \cap F \neq \emptyset \,\}$. If $|I| > 1$ the NFA is *multiple-entry*, otherwise it is *single-entry* (or *standard*). When $I = \{q_0\}$, we use $I = q_0$. If $|I| = 1$ and $|\delta(q, \sigma)| \leq 1$, for all $q \in Q, \sigma \in \Sigma$, $\boldsymbol{a}$ is *deterministic* (DFA). An NFA accepting a non-empty language is *trim* if every state is accessible from an initial state and every state leads to a final state. Two NFAs $\boldsymbol{a}$ and $\boldsymbol{a}'$ are *equivalent* if their languages are the same. An automaton $\boldsymbol{a}$ is equivalent to a regular expression $\boldsymbol{r}$ if $\mathcal{L}(\boldsymbol{a}) = \mathcal{L}(\boldsymbol{r})$. An NFA $\boldsymbol{a}$ is *minimal* if it has the smallest number of states among all NFAs that accept $\mathcal{L}(\boldsymbol{a})$. We can convert an NFA $\boldsymbol{a}$ into an equivalent DFA by the *determinisation* using the well-known subset construction.

The *left-quotient* of a language L by a word $w \in \Sigma^\star$ is the language

$$w^{-1}L = \{\, x \mid wx \in L \,\}.$$

Given two languages $L, R \subseteq \Sigma^\star$ the *left-quotient* of R with respect to (w.r.t.) L is

$$L^{-1}R = \{\, x \mid \exists u \in L \wedge ux \in R \,\} = \bigcup_{w \in L} w^{-1}R.$$

One can also say that $L^{-1}R$ is the *language quotient* of R by L and write

$$L^{-1}R = \{\, x \mid Lx \cap R \neq \emptyset \,\},$$

which implies the following.

Lemma 1. *$\varepsilon(L^{-1}R) = \varepsilon$ iff $L \cap R \neq \emptyset$.*

A language L induces in $\Sigma^\star$ the Nerode equivalence relation:

$$x \equiv_L y \iff x^{-1}L = y^{-1}L.$$

The language L is regular if and only if $\equiv_L$ has a finite number of equivalence classes, i.e., it has a finite number of left-quotients by a word [16]. This implies that a *minimal* DFA is unique up to the renaming of states. The *(deterministic) state complexity* of a language L, $\mathsf{sc}(L)$, is the size of its minimal DFA. The *nondeterministitic state complexity* of a language L, $\mathsf{nsc}(L)$, is defined analogously, where we assume that the NFAs are single-entry (standard).

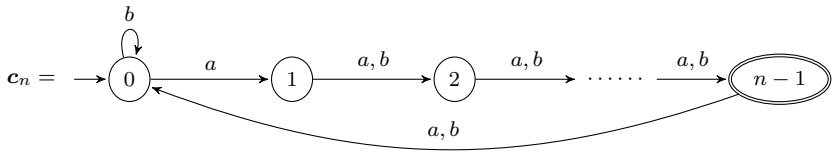


Fig. 1. DFAs $\boldsymbol{c}_n$, for $n \geq 1$, attaining the upper bound $2^n - 1$ for $\mathsf{sc}(L^{-1}R_n)$, where $R_n = \mathcal{L}(\boldsymbol{c}_n)$ and $L = \{a,b\}^\star$. We have that $R_n = \big(b + a(a+b)^{n-1}\big)^\star a(a+b)^{n-2} = b^\star a(a+b)^{n-2}\big((a+b)b^\star a(a+b)^{n-2}\big)^\star$.

3. Language Quotients and State Complexity

If R is a regular language and L is any language then $L^{-1}R$ is regular. A main result of [18] (quoted as Theorem 2 below) gives an attainable upper bound on the state complexity of the quotient operation between languages. Here, we show that the said upper bound of [18] can be improved as explained in Theorem 4. Incidentally, we show (non-surprisingly) in Lemma 5 that there is a stream of multi-entry NFAs such that an extra state is necessary to convert them to equivalent standard (single-entry) NFAs. We also show in Theorem 8 that computing the quotient $L^{-1}R$ of two regular languages given by NFAs can be done in quadratic time.

In [6], a *regular language stream* is a sequence $(R_n)_{n\geq n_0}$ of regular languages such that $\mathsf{sc}(R_n) = n$ for all n, where the index n starts at some position $n_0 \geq 1$. As this concept is tight to language complexity via *deterministic* automata we use here the term *deterministic regular stream* for $(R_n)_{n\geq n_0}$. If $\mathsf{nsc}(R_n)$ is super-polynomially smaller than the (deterministic) state complexity n, the stream $(R_n)_{n\geq n_0}$ is called *hard*, that is, when there is a polynomial f such that

$$\mathsf{nsc}(R_n) \leq f(\log n), \text{ for all } n \geq n_0.$$

We use the term *nondeterministic regular stream* for a sequence $(R_n)_{n\geq n_0}$ of regular languages such that $\mathsf{nsc}(R_n) = n$ for all $n \geq n_0$. We also introduce the term *multi-entry regular stream* for a sequence $(R_n)_{n\geq n_0}$ of regular languages such that each R_n is accepted by an n-state NFA with multiple initial states and n is the smallest number of states in such an NFA.

Theorem 2 (Yu et al. [18, Th. 4.1]) *Given a regular language R and any language L, we have that $\mathsf{sc}(L^{-1}R) \leq 2^{\mathsf{sc}(R)} - 1$. Moreover, this bound is attainable via the deterministic regular stream $(R_n)_{n\geq 1}$ in Fig. 1 and $L = \{a,b\}^\star$, that is, $\mathsf{sc}(L^{-1}R_n) = 2^n - 1$.*

Theorem 2 implies that $\mathsf{sc}(L^{-1}R_n) \leq 2^n - 1$, for any deterministic regular stream $(R_n)_{n\geq n_0}$. In Theorem 4 below, we show that the upper bound $2^n - 1$ can be revised as $2^{\mathsf{nsc}(R_n)} - 1$, which is still tight for the stream $(R_n)_{n\geq 1}$ in Fig. 1. If a deterministic regular stream $(R_n)_{n\geq n_0}$ is hard then the revised bound $2^{\mathsf{nsc}(R_n)} - 1$ can be much smaller than $2^n - 1$.

The following notation is based on the proof of Theorem 2 in [18].

Definition 3. *Let $\boldsymbol{a}$ be an NFA and let L be a language.*

(1) For any state p in $\boldsymbol{a}$, the NFA $\boldsymbol{a}_p$ is the same as $\boldsymbol{a}$ except that its set of final states is $\{p\}$.
(2) The set of states $S(\boldsymbol{a}, L)$ is defined as follows:

$$S(\boldsymbol{a}, L) = \{\, p \mid p \text{ is a state of } \boldsymbol{a} \text{ and } L \cap \mathcal{L}(\boldsymbol{a}_p) \neq \emptyset \,\}.$$

(3) The multi-entry NFA $\boldsymbol{a}_L$ results by making a copy of $\boldsymbol{a}$ that has $S(\boldsymbol{a}, L)$ as set of initial states—unless $S(\boldsymbol{a}, L) = \emptyset$, in which case $\boldsymbol{a}_L$ has only one state that is initial but not final, so $\boldsymbol{a}_L$ accepts $\emptyset$.

We have that $\mathcal{L}(\boldsymbol{a}_L) = L^{-1}\mathcal{L}(\boldsymbol{a})$ and $\boldsymbol{a}_L$ has no more states than $\boldsymbol{a}$ does.

Theorem 4. *Let R be a regular language and let L be any language.*

(1) $\mathsf{sc}(L^{-1}R) \leq 2^{\mathsf{nsc}(R)} - 1$ and this bound is attainable via the deterministic regular stream $(R_n)_{n\geq 1}$ in Fig. 1 and $L = \{a, b\}^\star$, that is, $\mathsf{sc}(L^{-1}R_n) = 2^{\mathsf{nsc}(R_n)} - 1$.
(2) There is a hard deterministic regular stream $(R_n)_{n\geq 4}$ such that $\mathsf{sc}(L^{-1}R_n) < 2^{(\log n)^2}$ and the alphabet of each R_n has cardinality $\leq 1 + \lfloor \log n \rfloor$.
(3) $\mathsf{nsc}(L^{-1}R) \leq \mathsf{nsc}(R)+1$ and this bound is attainable via a nondeterministic regular stream $(R_n)_{n\geq 2}$ and a language L, that is, $\mathsf{nsc}(L^{-1}R_n) = n + 1$.

For the last two statements in the above theorem we use a few intermediate results, presented next. The proof of the theorem is presented at the end of this subsection.

The first statement of the next lemma is folklore: converting a multi-entry NFA to an equivalent (standard) NFA can be done by introducing appropriately a single initial state. The second statement of the lemma shows that there are streams of multi-entry NFAs such that an extra state is necessary to convert them to equivalent NFAs.

Lemma 5. *Let $n \geq 1$. For every multi-entry NFA $\boldsymbol{a}_n$ with n states, there is an equivalent NFA having $n+1$ states; hence, $\mathsf{nsc}(\mathcal{L}(\boldsymbol{a}_n)) \leq n+1$. Moreover, this bound is attainable via a multi-entry regular stream $(Q_n)_{n\geq 2}$ such that $\mathsf{nsc}(Q_n) = n + 1$ and the alphabet of each Q_n is $\{a, b\}$.*

Proof. For the first statement, let I be the set of initial states, let K be the set of states, and let Σ be the alphabet of $\boldsymbol{a}_n$. We make a copy $\boldsymbol{a}'_n$ of $\boldsymbol{a}_n$ adding to it a new state s as its only initial state, as well as the transitions (s, σ, q) for all transitions (p, σ, q) of $\boldsymbol{a}_n$ with $p \in I$, $q \in K$ and $\sigma \in \Sigma$. Then $\mathcal{L}(\boldsymbol{a}'_n) = \mathcal{L}(\boldsymbol{a}_n)$ and $\boldsymbol{a}'_n$ has $n + 1$ states, as required.

For the second statement, for the alphabet $\Sigma = \{a, b\}$ and for $n \geq 2$, consider the language

$$Q_n = a\Sigma^{n-2}(\Sigma a \Sigma^{n-2})^\star \bigcup \Sigma^{n-2}(\Sigma a \Sigma^{n-2})^\star.$$

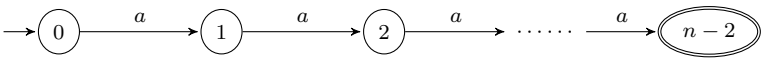
We need to show that $(Q_n)_{n\geq 2}$ is a multi-entry regular stream and that $\mathsf{nsc}(Q_n) = n+1$. As the shortest word of Q_n is of length $n-2$, we have that $\mathsf{nsc}(Q_n) \geq n-1$. As each Q_n can be accepted by a two-initial state NFA with n states, we have that $(Q_n)_{n\geq 2}$ is indeed a multi-entry regular stream and that $\mathsf{nsc}(Q_n) \leq n+1$. To prove that $\mathsf{nsc}(Q_n) = n+1$, we need to exclude the case of $\mathsf{nsc}(Q_n) \in \{n-1, n\}$.

First note that:

(i) the *valid word lengths of* Q_n are $(n-1)+in$ and $(n-2)+in$, for all $i \geq 0$;
(ii) $a^{n-2} \in Q_n$ and a^{n-2} is a shortest word of Q_n.

Consider any NFA $\boldsymbol{b}_n$ accepting Q_n. We have the following statements about $\boldsymbol{b}_n$:

(A) There must be states $0, 1, \ldots, (n-2)$ in $\boldsymbol{b}_n$ and transitions as shown below:



(B) None of the states $0, 1, \ldots, (n-3)$ is final, else $\boldsymbol{b}_n$ would accept a word of length $< n-2$; and no loop transition (p, σ, p) can exist, with $0 \leq p \leq n-2$, else, $\boldsymbol{b}_n$ would accept a word of invalid length n.

(C) $\boldsymbol{b}_n$ has no (backward) transition (q, σ, p) with $0 \leq p < q \leq (n-2)$, else, $\boldsymbol{b}_n$ would accept a word of invalid length $\ell = q+1+(n-2)-p = q+n-1-p$, corresponding to a path from 0 to q to p to $(n-2)$: $n \leq \ell \leq n+(n-3)$.

(D) $\boldsymbol{b}_n$ has no transition (p, σ, q) with $q > p+1$, else $\boldsymbol{b}_n$ would accept a word of length $< n-2$.

Thus, using only the above $n-1$ states the NFA $\boldsymbol{b}_n$ cannot accept any words of length $> n-2$. *Thus,* $\boldsymbol{b}$ *must have another state* $(n-1)$*; hence,* $\mathsf{nsc}(Q_n) \geq n$. We show next that $\boldsymbol{b}_n$ *cannot have exactly the* n *states* $0, 1, \ldots, (n-1)$. Let

$$p = \min\{\, i \mid 0 \leq i \leq (n-2) \text{ and } \exists \text{ transition } (i, \sigma, n-1) \text{ in } \boldsymbol{b}_n \,\}.$$

We have:

(E) If $p \leq n-3$ then there is no transition from $(n-1)$ to some q with $q \leq p$—else, $\boldsymbol{b}_n$ would accept a word of invalid length $\ell = p+1+1+(n-2)-q = p+n-q$, corresponding to a path from 0 to p, to $(n-1)$, to q, to $(n-2)$: $n \leq \ell \leq n+(n-3)$.

We consider two cases.

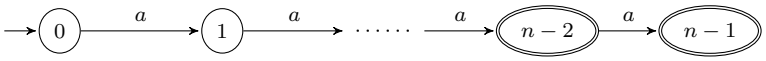
State $(n-1)$ is not final: First we have that $p \leq n-3$; else, $p = n-2$ implies that $\boldsymbol{b}_n$ cannot accept a word of length $n-1$. As $\boldsymbol{b}_n$ accepts a^{n-1} there must be a transition $(n-1, a, q)$ for some *smallest* $q \leq n-2$. By (E), we have that $p+1 \leq q \leq n-2$. Also, as Q_n is infinite, there must be a state r, with $q \leq r \leq n-2$, with a transition $(r, \sigma, n-1)$ going back to $(n-1)$. Then, $\boldsymbol{b}_n$ accepts a word of

invalid length $\ell = r + 1 + 1 + (n-2) - q = r + n - q$, corresponding to a path from 0 to r to $(n-1)$ to q to $(n-2)$: $n \le \ell \le n + (n-3)$. Hence, state $(n-1)$ must be final.

State $(n-1)$ is final: Then, we must have $p = n-2$.

This is why: As $\boldsymbol{b}_n$ has a transition $(p, \sigma, n-1)$ and accepts no words shorter than $(n-2)$, we have that $p \in \{n-3, n-2\}$. If $p = n-3$ then (E) implies that there is no transition from $(n-1)$ to any state $i \le n-3$. As Q_n is infinite, $(n-1)$ must have a transition to $(n-2)$ and also $(n-2)$ must have a transition to $(n-1)$, violating the possible lengths of that accepted words.

As a^{n-1} must be accepted, $\boldsymbol{b}_n$ should include the following pattern with the constraints (A)–(E):



We also have:

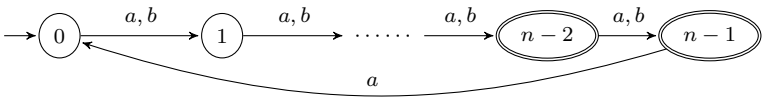
(F) To accept Σ^{n-2}, each of the first $(n-2)$ arrows shown above must include the label b. In fact, to also accept $a^{n-2}b$, the last arrow shown above must also include the label b.

(G) $\boldsymbol{b}_n$ cannot have a transition $(n-1, \sigma, i)$ with $1 \le i \le n-2$, as $\boldsymbol{b}_n$ would accept a word of invalid length between n and $n + (n-3)$.

(H) Similarly $\boldsymbol{b}_n$ cannot have a transition $(n-1, \sigma, n-1)$.

(I) As Q_n is infinite, $\boldsymbol{b}_n$ must have a transition $(n-1, \sigma, 0)$. In fact this transition cannot be $(n-1, b, 0)$ because $\boldsymbol{b}_n$ would accept $ba^{n-3}aba^{n-2} \notin Q_n$.

Thus, $\boldsymbol{b}_n$ must look as follows:



This implies that $\boldsymbol{b}_n$ accepts the word $ab^{n-2}aba^{n-2} = ab^{n-3}baba^{n-3}a \notin Q_n$. Hence, state $(n-1)$ cannot be final either and $\boldsymbol{b}_n$ cannot have exactly n states. □

The next lemma is based on the fact, [10], that for any alphabet Σ, any $\sigma \in \Sigma$, and any $m \ge 1$, the minimal DFA accepting the language $\Sigma^\star \sigma \Sigma^{m-1}$ has 2^m states.

Lemma 6 (Crochemore et al. [10]) *For each $p = 1, 2, \ldots$, the DFA $\boldsymbol{d}_p$ in Fig. 2 is minimal, has 2^p states, and accepts the language $A_p^\star p A_p^{p-1}$ over the alphabet $A_p = \{0, p\}$. Moreover, there is a $(p+1)$-state NFA accepting the same language.*

Lemma 7. *There is a deterministic regular stream $(R_n)_{n \ge 4}$ such that $\mathsf{nsc}(R_n) \le 2 + \sum_{p=1}^{\lfloor \log n \rfloor} p$ and the alphabet of each R_n has cardinality $\le 1 + \lfloor \log n \rfloor$.*

Proof. For each $n \ge 4$, we make a minimal DFA $\boldsymbol{m}$ of n states using as components the DFAs $\boldsymbol{d}_p$ in Lemma 6. Each $\boldsymbol{d}_p$ has 2^p states, so we are going to use a $\boldsymbol{d}_p$ for each non-zero bit position p in the binary representation of $N = n-2$ or $n-3$ depending on whether or not n is even. First, consider the case where n is even and

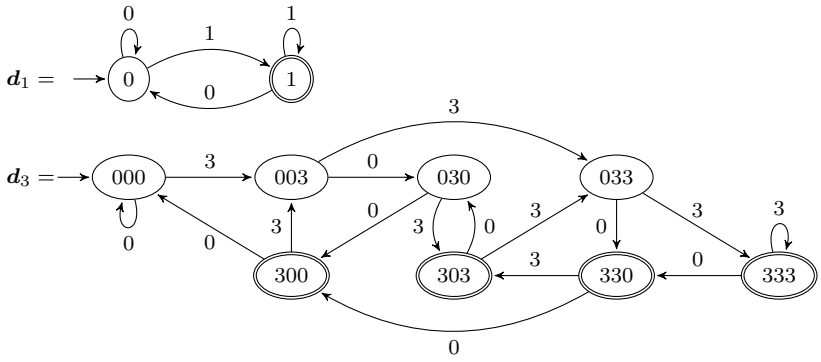


Fig. 2. Minimal DFAs $\boldsymbol{d}_p$, for $p = 1, 2, \ldots$. Each $\boldsymbol{d}_p$ has alphabet $A_p = \{0, p\}$ and accepts the language $A_p^\star\, pA_p^{p-1}$ = all words whose p-th symbol from the end is p. Each DFA $\boldsymbol{d}_p$ has 2^p states, each of the form $c_1 \cdots c_p$, with $c_i \in \{0, p\}$. Each state remembers the last p symbols that have been read so far.

$n \geq 4$. Let $N = n - 2$, which is still even. Define the set $P(N) =$ the positions of the non-zero bits in the (minimal) binary representation of N, where position 0 is the rightmost one and position $\lfloor \log N \rfloor$ is the leftmost one. For example, if $N = 12$ then $P(12) = \{2, 3\}$. Obviously,

$$N = \sum_{p \in P(N)} 2^p.$$

The required language R_n is

$$R_n = \bigcup_{p \in P(N)} pA_p^*\, pA_p^{p-1}$$

and is accepted by the minimal DFA $\boldsymbol{m}$ such that (i) $\boldsymbol{m}$ has $N + 2$ states and alphabet $\Sigma = \{0\} \cup P(N)$; (ii) for each $p \in P(N)$, the DFA $\boldsymbol{d}_p$ in Lemma 6 is a component of $\boldsymbol{m}$; (iii) the start state s of $\boldsymbol{m}$ has a transition (s, p, s_p) to the start state s_p of each $\boldsymbol{d}_p$; (iv) the final states of $\boldsymbol{m}$ are all the final states in the $\boldsymbol{d}_p$'s; (v) $\boldsymbol{m}$ has a sink state. Recall, that each $\boldsymbol{d}_p$ is minimal, which implies that its states in $\boldsymbol{m}$ are non-equivalent. Moreover, for any p, q with $p \neq q$ we have that no two states $b_1 \cdots b_p$ and $c_1 \cdots c_q$ are equivalent as their transitions involve different alphabets.

The DFA $\boldsymbol{m}$ has $n = N + 2$ states; hence $\mathsf{sc}(R_n) = n$. We also need to show that $\mathsf{nsc}(R_n) \leq 2 + \sum_{i=1}^{\lfloor \log n \rfloor} i$. By Lemma 6, each language $A_p^*\, pA_p^{p-1}$ can be accepted by an NFA of $(p + 1)$-states. Thus, the language $R_n = \mathcal{L}(\boldsymbol{m})$ can be accepted by an NFA with

$$1 + \sum_{p \in P(N)} (p + 1) \tag{2}$$

states.

For the case where $n = \text{odd} \geq 5$, we let $N = n - 3$, which is even. The required minimal DFA $\boldsymbol{m}$ is the same as above plus a new final state 0 and new transitions $(s, 0, 0)$ and $(0, \sigma, \text{sink})$, for all $\sigma \in \Sigma$. Thus, $\boldsymbol{m}$ accepts the same language as the $\boldsymbol{m}$ for $n - 1$ plus the word 0. The language R_n can be accepted by an NFA with a number of states equal to 1 plus the value in (2). □

Proof of Theorem 4. For the first statement, we repeat the construction of [18] that was given for the upper bound $2^{\mathsf{sc}(R)} - 1$, by using an NFA for R, instead of a DFA. Specifically, let $m = \mathsf{nsc}(R)$ and let $\boldsymbol{a}$ be an m-state NFA accepting R. Consider the set of states $S(\boldsymbol{a}, L)$ and the multi-entry NFA $\boldsymbol{a}_L$ accepting $L^{-1}R$, which has $\leq m$ states (see Def. 3). Using the subset construction on $\boldsymbol{a}_L$ with $S(\boldsymbol{a}, L)$ as the start state, we get a DFA accepting $L^{-1}R$, which has at most $2^m - 1$ states (as in [18], the minimal DFA for $L^{-1}R$ has no state $\emptyset$). Hence, $\mathsf{sc}(L^{-1}R) \leq 2^{\mathsf{nsc}(R)} - 1$, as required. We also need to show that the revised upper bound $2^{\mathsf{nsc}(R)} - 1$ is attainable. The DFAs $\boldsymbol{c}_n$ in Fig. 1 are from [18]. Each language $R_n = \mathcal{L}(\boldsymbol{c}_n)$ has $\mathsf{sc}(R_n) = \mathsf{nsc}(R_n) = n$ because the shortest word of R_n is of length $n - 1$. Then, trivially, we have that the same language stream $(R_n)_{n \geq 1}$ attains the revised bound $2^{\mathsf{nsc}(R_n)} - 1$, as required.

Second statement: the upper bound $2^{(\log n)^2}$ follows from the first statement and from Lemma 7. This implies that the stream of Lemma 7 is hard.

Third statement: as in the proof of the first statement, consider an NFA $\boldsymbol{a}$ with $m = \mathsf{nsc}(R)$ states and the multi-entry NFA $\boldsymbol{a}_L$ that accepts $L^{-1}R$ and has at most m states. Convert $\boldsymbol{a}_L$ to an equivalent standard (single entry) NFA $\boldsymbol{a}'_L$ using Lemma 5. As $\boldsymbol{a}'_L$ has at most $1 + m$ states, we have that $\mathsf{nsc}(L^{-1}R) \leq \mathsf{nsc}(R) + 1$, as required. To show that the bound $\mathsf{nsc}(R) + 1$ is tight, we notice that in the proof of Lemma 5, each $Q_n = L^{-1}R_n$, where $L = \{\varepsilon, a\}$ and

$$R_n = a\Sigma^{n-2}\big(\Sigma a \Sigma^{n-2}\big)^\star.$$

As the shortest word in R_n is of length $n-1$ and there is an n-state NFA $\boldsymbol{a}_n$ accepting it, we have that $\mathsf{nsc}(R_n) = n$, so the stream $(R_n)_{n \geq 2}$ is a nondeterministic regular stream. Then for the multi-entry NFA $(\boldsymbol{a}_n)_L$ (= copy of $\boldsymbol{a}_n$ with two initial states) accepting $L^{-1}R_n$, we have that $\mathsf{nsc}(L^{-1}R_n) = n + 1 = \mathsf{nsc}(R_n) + 1$, as required. □

3.1. *Computing Language Quotients using NFAs*

As stated in [18], computing an automaton for $L^{-1}R$, for given regular R, depends on how L is described. If both L, R are described via NFAs $\boldsymbol{b}, \boldsymbol{a}$, respectively, then the computation of the NFA accepting $L^{-1}R$ boils down to computing the following set of states of $\boldsymbol{a}$

$$S(\boldsymbol{a}, L) = \{\, p \mid p \text{ is a state on } \boldsymbol{a} \text{ and } L \cap \mathcal{L}(\boldsymbol{a}_p) \neq \emptyset \,\}$$

(see Def. 3). Computing $S(\boldsymbol{a}, L)$ can be done as follows: for each state p of $\boldsymbol{a}$, use the standard product construction between $\boldsymbol{a}_p$ and $\boldsymbol{b}$ to get an NFA accepting $L \cap \mathcal{L}(\boldsymbol{a}_p)$

and add p to $S(\boldsymbol{a}, L)$ if $L \cap \mathcal{L}(\boldsymbol{a}_p) \neq \emptyset$. This process requires time $O(k_{\boldsymbol{a}} \cdot |\boldsymbol{a}| \cdot |\boldsymbol{b}|)$, where $k_{\boldsymbol{a}}$ is the number of states in $\boldsymbol{a}$. The proof of the next theorem shows that we can determine the desired set S in time $O(|\boldsymbol{a}| \cdot |\boldsymbol{b}|)$ using only one product construction.

Theorem 8. *There is an algorithm that computes, for given NFAs $\boldsymbol{a}, \boldsymbol{b}$, an NFA of size $O(|\boldsymbol{a}|)$ accepting the quotient $\mathcal{L}(\boldsymbol{b})^{-1}\mathcal{L}(\boldsymbol{a})$. The algorithm works in time $O(|\boldsymbol{a}| \cdot |\boldsymbol{b}|)$.*

Proof. As in the proof of Theorem 4.3, the algorithm computes the set $S(\boldsymbol{a}, L)$ of those states p of $\boldsymbol{a}$ such that $\mathcal{L}(\boldsymbol{b}) \cap \mathcal{L}(\boldsymbol{a}_p) \neq \emptyset$. The version of $\boldsymbol{a}$ with set of initial states $S(\boldsymbol{a}, L)$ is the multi-entry NFA $\boldsymbol{a}_L$ accepting the desired quotient, and the required standard (single-entry) NFA is the NFA resulting from $\boldsymbol{a}_L$ using the folklore method of Lemma 5. The size of the computed NFA is $O(|\boldsymbol{a}|)$, but its computation requires computing the above set of states $S(\boldsymbol{a}, L)$. The algorithm computes $S(\boldsymbol{a}, L)$ in two steps. First it constructs the product machine $\boldsymbol{a} \times \boldsymbol{b}$: if (p_1, σ, q_1) and (p_2, σ, q_2) are transitions in $\boldsymbol{a}, \boldsymbol{b}$ then $\big((p_1, q_1), \sigma, (p_2, q_2)\big)$ is a transition of $\boldsymbol{a} \times \boldsymbol{b}$. Let s_1, s_2 be the initial states of $\boldsymbol{a}, \boldsymbol{b}$, respectively. The second step is to compute the set

$$I = \{\, p \mid \exists \text{ final state } g \text{ in } \boldsymbol{b} \text{ such that } (p, g) \text{ is reachable from } (s_1, s_2) \text{ in } \boldsymbol{a} \times \boldsymbol{b} \,\}.$$

One verifies that the desired set $S(\boldsymbol{a}, L)$ is equal to the set I. □

4. Automata and Expressions for Language Quotients based on Derivatives

In this section, we consider language quotients of languages represented by regular expressions and derivative based tools for their manipulation. In Section 4.1, we recall the basic notion of derivative of regular expression by a word due to Brzozowski [7], and we present an algorithm that uses derivatives to compute, given two regular expressions $\boldsymbol{t}$ and $\boldsymbol{r}$, a regular expression $\boldsymbol{s}$ that represents $\mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$. In Section 4.2, we recall the definition of *partial* derivatives of a regular expression by a word, and we give a partial-derivative version of the previous algorithm, which avoids the use of expressions modulo some equivalence rules. In Section 4.3, we extend regular expressions with the quotient operation and define partial derivatives for expressions involving the quotient operator, which leads to the construction of the partial derivative automaton for such expressions.

4.1. *Computing Language Quotients via Derivatives*

The *derivative* by $\sigma \in \Sigma$ of a regular expression $\boldsymbol{r}$ is a regular expression $\frac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r})$, inductively defined on the structure or $\boldsymbol{r}$ by

$$\begin{aligned} &\tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\emptyset) = \tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\varepsilon) = \emptyset, && \tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r}+\boldsymbol{r}') = \tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r}) + \tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r}'), \\ &\tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\sigma') = \begin{cases} \varepsilon & \text{if } \sigma' = \sigma, \\ \emptyset & \text{otherwise,} \end{cases} && \tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r}\boldsymbol{r}') = \tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r})\,\boldsymbol{r}' + \varepsilon(\boldsymbol{r})\tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r}'), \\ & && \tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r}^\star) = \tfrac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r})\,\boldsymbol{r}^\star. \end{aligned} \tag{3}$$

The *derivative by a word* of a regular expression $\boldsymbol{r}$ is inductively defined on the structure of a word by, $\frac{\mathrm{d}}{\mathrm{d}\,\varepsilon}(\boldsymbol{r}) = \boldsymbol{r}$ and $\frac{\mathrm{d}}{\mathrm{d}\,w\sigma}(\boldsymbol{r}) = \frac{\mathrm{d}}{\mathrm{d}\,\sigma}\big(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\big)$, for $w \in \Sigma^\star$. Given a word $w \in \Sigma^\star$, the language of $\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})$ is the left quotient of $\mathcal{L}(\boldsymbol{r})$ by w, i.e.,

$$\mathcal{L}\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\right) = \{\, x \mid wx \in \mathcal{L}(\boldsymbol{r}) \,\} = w^{-1}\mathcal{L}(\boldsymbol{r}).$$

Lemma 9 (Brzozowski [7, Th. 4.4-4.5]) *For every $\boldsymbol{r} \in RE$, $\sigma \in \Sigma$, and $w \in \Sigma^\star$, one has*

$$\boldsymbol{r} \doteq \varepsilon(\boldsymbol{r}) + \sum_{\sigma\in\Sigma} \sigma \frac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r}), \tag{4}$$

$$\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r}) \doteq \varepsilon\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\right) + \sum_{\sigma\in\Sigma} \sigma \frac{\mathrm{d}}{\mathrm{d}\,w\sigma}(\boldsymbol{r}). \tag{5}$$

Moreover, $w \in \mathcal{L}(\boldsymbol{r})$ if and only if $\varepsilon\big(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\big) = \varepsilon$, i.e., $\varepsilon \in \mathcal{L}\big(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\big)$.

The set of all derivatives of $\boldsymbol{r}$, $\{\, \frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r}) \mid w \in \Sigma^\star \,\}$, may not be finite. For finiteness, Brzozowski [7] considered the quotient of that set modulo some regular expression equivalences, namely the associativity, commutativity, identity element, and idempotency of $+$ (ACI rules); and the following rules:

$$\begin{aligned} &\boldsymbol{r}\varepsilon \doteq \varepsilon\boldsymbol{r} \doteq \boldsymbol{r}, \\ &\emptyset\boldsymbol{r} \doteq \boldsymbol{r}\emptyset \doteq \emptyset. \end{aligned} \tag{6}$$

The equivalence class of a derivative $\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})$ (under the above rules) has a unique representative that we denote by $[\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})]$, for $w \in \Sigma^\star$. The set $\mathcal{D}(\boldsymbol{r})$ of all equivalence classes (under the above rules) is called the set of *dissimilar* derivatives. The *Brzozowski automaton* for $\boldsymbol{r}$ is the DFA defined as follows:

$$\boldsymbol{a}_{\mathrm{B}}(\boldsymbol{r}) = \langle \mathcal{D}(\boldsymbol{r}), \Sigma, \delta_{\mathrm{B}}, [\boldsymbol{r}], F_{\mathrm{B}} \rangle,$$

where $F_{\mathrm{B}} = \{\, [\boldsymbol{r}'] \in \mathcal{D}(\boldsymbol{r}) \mid \varepsilon(\boldsymbol{r}') = \varepsilon \,\}$, and $\delta_{\mathrm{B}}([\boldsymbol{r}'], \sigma) = [\frac{\mathrm{d}}{\mathrm{d}\,\sigma}(\boldsymbol{r}')]$, for all $[\boldsymbol{r}'] \in \mathcal{D}(\boldsymbol{r})$ and $\sigma \in \Sigma$.

Proposition 10 (Brzozowski [7, Th. 5.1]) $\mathcal{L}(\boldsymbol{r}) = \mathcal{L}(\boldsymbol{a}_{\mathrm{B}}(\boldsymbol{r}))$.

In the following, we will identify derivatives with their equivalence class representative and omit the brackets ($[\cdot]$). The language of a derivative is a unique left-quotient by a word, but two derivatives may represent the same language. Known upper bounds for the state complexity of several operations can be obtained using derivatives by tightening the bounds of $|\mathcal{D}(\boldsymbol{r})|$ [5]. For that the following proposition was useful.

Proposition 11 (Brzosowski [7, Th. 4.3], [5, Th. 1]) *If $\boldsymbol{r}$ and $\boldsymbol{r}'$ are regular expressions and $w \in \Sigma^\star$ then*

$$\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r}+\boldsymbol{r}') \doteq \frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r}) + \frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r}'), \tag{7}$$

$$\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r}\boldsymbol{r}') \doteq \frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\,\boldsymbol{r}' + \varepsilon(\boldsymbol{r})\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r}') + \sum_{\substack{w=uv\\ u,v\in\Sigma^+}} \varepsilon\left(\frac{\mathrm{d}}{\mathrm{d}\,u}(\boldsymbol{r})\right)\frac{\mathrm{d}}{\mathrm{d}\,v}(\boldsymbol{r}'). \tag{8}$$

We have $\frac{\mathrm{d}}{\mathrm{d}\,\varepsilon}(\boldsymbol{r}^\star) \doteq \varepsilon + \boldsymbol{r}\boldsymbol{r}^\star$ and if $w \in \Sigma^+$, then

$$\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r}^\star) \doteq \left(\sum_{\substack{w=uv\\ u,v\in\Sigma^\star}} \varepsilon\left(\frac{\mathrm{d}}{\mathrm{d}\,u}(\boldsymbol{r}^\star)\right)\frac{\mathrm{d}}{\mathrm{d}\,v}(\boldsymbol{r})\right)\boldsymbol{r}^\star. \tag{9}$$

Now, we can use derivatives by a word to compute language quotients. Given two regular expressions $\boldsymbol{t}$ and $\boldsymbol{r}$, we have

Lemma 12. $\mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r}) = \bigcup_{w\in\Sigma^\star} \varepsilon\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{t})\right)\mathcal{L}\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\right)$.

Proof. Let $x \in \mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$. Then, there exists $w \in \mathcal{L}(\boldsymbol{t})$, such that $wx \in \mathcal{L}(\boldsymbol{r})$. This means that $\varepsilon\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{t})\right) = \varepsilon$ and $x \in \mathcal{L}\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\right)$. Analogously, we obtain that $\bigcup_{w\in\Sigma^\star} \varepsilon\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{t})\right)\mathcal{L}\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\right) \subseteq \mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$. □

Using Lemma 12 and the finiteness of the set of all derivatives of a regular expression by word, $\mathcal{D}(\cdot)$, we can define an (iterative) algorithm that computes directly a standard regular expression that represents $\mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$. Algorithm 1 is an adaption of the one for deciding the equivalence of two regular expressions using derivatives [1,15]. The algorithm computes successively and breadth-first, the derivatives of $\boldsymbol{t}$ and $\boldsymbol{r}$ by words generated in lexicographical order and stores them in a set S. For each pair $\left([\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{t})], [\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})]\right) \in S$, one tests if $w \in \mathcal{L}(\boldsymbol{t})$. If that holds, $[\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})]$ will be one of the disjuncts ($+$) of a (standard) regular expression representing $\mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$. To ensure the finiteness of the process, the algorithm explicitly uses the equivalence class of each derivative [a] and keeps the pairs already tackled in a set H. From this, we obtain the following

[a]Considering the ACI rules and the ones in Eq. (6).

Theorem 13. *Given regular expressions $\boldsymbol{t}$ and $\boldsymbol{r}$, Algorithm 1 computes a regular expression that represents $\mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$.*

Proof. Termination follows from the finiteness of the sets $\mathcal{D}(\boldsymbol{r})$ and $\mathcal{D}(\boldsymbol{t})$. For the correctness, if $x \in \mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$ by Lemma 12 there exists $w \in \mathcal{L}(\boldsymbol{t})$ such that $\varepsilon\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{t})\right) = \varepsilon$ and $x \in \mathcal{L}\left(\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})\right)$. However, there exist $\boldsymbol{t}'$ and $\boldsymbol{r}'$ such that $\boldsymbol{t}' = [\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{t})]$ and $\boldsymbol{r}' = [\frac{\mathrm{d}}{\mathrm{d}\,w}(\boldsymbol{r})]$ and $(\boldsymbol{t}', \boldsymbol{r}') \in H$. Then, $\boldsymbol{r}'$ is a summand of $\boldsymbol{s}$ and $x \in \mathcal{L}(\boldsymbol{s})$. If $x \in \mathcal{L}(\boldsymbol{s})$, then also $x \in \mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$. Note that the invariant of the **while** loop satisfies $\mathcal{L}(\boldsymbol{s}) \subseteq \mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$. □

Algorithm 1 Quotient of two regular expressions. Returns a regular expression.

```
1: function QUOTIENT(t ,r)
2:     S ← {(t, r)}
3:     H ← ∅
4:     s ← ∅
5:     while (t, r) ←POP(S) do
6:         if ε(t) = ε then
7:             s ← s + r
8:         PUSH(H, (t, r))
9:         for σ ∈ Σ do
10:            (t′, r′) ← ([d/dσ (t)], [d/dσ (r)])
11:            if (t′, r′) ∉ H then
12:                PUSH(S,(t′, r′))
       return s
```

Example 14. *Let $\boldsymbol{r} = (a^\star b + bb^\star a)^\star$ and $\boldsymbol{t} = (aa + bb)^\star$. Then applying Algorithm 1, one has the pairs $(\boldsymbol{t}, \boldsymbol{r})$; $(\boldsymbol{t}_1, \boldsymbol{r}_1)$ with $\boldsymbol{t}_1 = \frac{\mathrm{d}}{\mathrm{d}\,a}(\boldsymbol{t}) = a\boldsymbol{t}$ and $\boldsymbol{r}_1 = \frac{\mathrm{d}}{\mathrm{d}\,a}(\boldsymbol{r}) = (a^\star b)\boldsymbol{r}$; $(\boldsymbol{t}_2, \boldsymbol{r}_2)$ with $\boldsymbol{t}_2 = \frac{\mathrm{d}}{\mathrm{d}\,b}(\boldsymbol{t}) = b\boldsymbol{t}$ and $\boldsymbol{r}_2 = \frac{\mathrm{d}}{\mathrm{d}\,b}(\boldsymbol{r}) = (\varepsilon + b^\star a)\boldsymbol{r}$; $(\boldsymbol{t}_3, \boldsymbol{r}_3)$ with $\boldsymbol{t}_3 = \frac{\mathrm{d}}{\mathrm{d}\,a}(\boldsymbol{t}_1) = \boldsymbol{t}$ and $\boldsymbol{r}_3 = \frac{\mathrm{d}}{\mathrm{d}\,a}(\boldsymbol{r}_1) = \boldsymbol{r}_1$; $(\boldsymbol{t}_4, \boldsymbol{r}_4)$ with $\boldsymbol{t}_4 = \frac{\mathrm{d}}{\mathrm{d}\,b}(\boldsymbol{t}_1) = \emptyset$ and $\boldsymbol{r}_4 = \frac{\mathrm{d}}{\mathrm{d}\,b}(\boldsymbol{r}_1) = \boldsymbol{r}$; $(\boldsymbol{t}_5, \boldsymbol{r}_5)$ with $\boldsymbol{t}_5 = \frac{\mathrm{d}}{\mathrm{d}\,a}(\boldsymbol{t}_2) = \emptyset$ and $\boldsymbol{r}_5 = \frac{\mathrm{d}}{\mathrm{d}\,a}(\boldsymbol{r}_2) = \boldsymbol{r}_2$; $(\boldsymbol{t}_6, \boldsymbol{r}_6)$ with $\boldsymbol{t}_6 = \frac{\mathrm{d}}{\mathrm{d}\,b}(\boldsymbol{t}_2) = \boldsymbol{t}$ and $\boldsymbol{r}_6 = \frac{\mathrm{d}}{\mathrm{d}\,b}(\boldsymbol{r}_2) = \boldsymbol{r}_2$. Considering the cases where $\varepsilon(\boldsymbol{t}_i) = \varepsilon$ we have $\boldsymbol{s} = \boldsymbol{r} + \boldsymbol{r}_1 + \boldsymbol{r}_2$. The remaning pairs added to S before termination do not add more summands to $\boldsymbol{s}$, thus $\mathcal{L}(\boldsymbol{s}) = \mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$.*

4.2. *Computing Language Quotients via Partial Derivatives*

The partial derivative automaton of a regular expression was introduced independently by Mirkin [14] and Antimirov [2] as a nondeterministic counterpart of Brzozowski automaton. Champarnaud and Ziadi [8] proved that the two formulations are equivalent.

Given a regular expression $\boldsymbol{r}$ and a symbol σ, the language of the set of partial derivatives $\partial_\sigma(\boldsymbol{r})$ is $\sigma^{-1}\mathcal{L}(\boldsymbol{r})$. However, partial derivatives are more convenient to use than derivatives as the set of partial derivatives by all words of an expression is finite (and not only finite modulo ACI and other rules).

For a regular expression $\boldsymbol{r} \in RE$ and a symbol $\sigma \in \Sigma$, the *set of partial derivatives by σ of $\boldsymbol{r}$* is the set of regular expressions defined inductively as follows:

$$\begin{array}{ll} \frac{\partial}{\partial \sigma}(\emptyset) = \frac{\partial}{\partial \sigma}(\varepsilon) = \emptyset, & \frac{\partial}{\partial \sigma}(\boldsymbol{r}+\boldsymbol{r}') = \frac{\partial}{\partial \sigma}(\boldsymbol{r}) \cup \frac{\partial}{\partial \sigma}(\boldsymbol{r}'), \\ \frac{\partial}{\partial \sigma}(\sigma') = \begin{cases} \{\varepsilon\} & \text{if } \sigma' = \sigma, \\ \emptyset & \text{otherwise,} \end{cases} & \frac{\partial}{\partial \sigma}(\boldsymbol{r}\boldsymbol{r}') = \frac{\partial}{\partial \sigma}(\boldsymbol{r})\,\boldsymbol{r}' \cup \varepsilon(\boldsymbol{r})\frac{\partial}{\partial \sigma}(\boldsymbol{r}'), \\ & \frac{\partial}{\partial \sigma}(\boldsymbol{r}^\star) = \frac{\partial}{\partial \sigma}(\boldsymbol{r})\,\boldsymbol{r}^\star, \end{array} \tag{10}$$

where for $S \subseteq RE$ and $\boldsymbol{r}' \in RE$, we define $S\emptyset = \emptyset S = \emptyset$, $S\varepsilon = \varepsilon S = S$, and $S\boldsymbol{r}' = \{\, \boldsymbol{r}\boldsymbol{r}' \mid \boldsymbol{r} \in S\}$, where if $\boldsymbol{r} \in \{\emptyset, \varepsilon\}$ the identities Eq. (6) are used. With these definitions, the above equations ensure that the result is a set of expressions (and not one expression as in the case of derivatives). For instance, for the concatenation we obtain a set of expressions of the form $\boldsymbol{s}\boldsymbol{r}'$ with $\boldsymbol{s} \in \frac{\partial}{\partial \sigma}(\boldsymbol{r})$ or just $\boldsymbol{s} \in \frac{\partial}{\partial \sigma}(\boldsymbol{r}')$. Moreover, we emphasize that the symbol $\emptyset$ used in Equations (10) denotes the empty set, whereas in Equations (3), $\emptyset$ is the regular expression that represents the empty set.

The definition of partial derivatives by one symbol can be extended in a natural way to partial derivatives of sets of regular expressions and to partial derivatives by words. In particular, if $S \subseteq RE$, $\frac{\partial}{\partial \sigma}(S) = \bigcup_{\boldsymbol{s} \in S} \frac{\partial}{\partial \sigma}(\boldsymbol{s})$ and the set of partial derivatives by $w \in \Sigma^\star$ of $\boldsymbol{r} \in RE$ is inductively defined by $\frac{\partial}{\partial \varepsilon}(\boldsymbol{r}) = \{\boldsymbol{r}\}$ and $\frac{\partial}{\partial w\sigma}(\boldsymbol{r}) = \frac{\partial}{\partial \sigma}\left(\frac{\partial}{\partial w}(\boldsymbol{r})\right)$. For $w \in \Sigma^\star$, we have

$$\mathcal{L}\left(\frac{\partial}{\partial w}(\boldsymbol{r})\right) = \bigcup_{\boldsymbol{r}' \in \frac{\partial}{\partial w}(\boldsymbol{r})} \mathcal{L}(\boldsymbol{r}') = \mathcal{L}\left(\frac{\mathrm{d}}{\mathrm{d}\, w}(\boldsymbol{r})\right) = w^{-1}\mathcal{L}(\boldsymbol{r}). \tag{11}$$

The set of all partial derivatives of $\boldsymbol{r}$ by nonempty words is

$$\partial^+(\boldsymbol{r}) = \bigcup_{w \in \Sigma^\star \setminus \{\varepsilon\}} \frac{\partial}{\partial w}(\boldsymbol{r}).$$

Eq. (4) of Lemma 9 can be redefined as follows:

$$\boldsymbol{r} \doteq \varepsilon(\boldsymbol{r}) \cup \bigcup_{\sigma \in \Sigma} \sigma \frac{\partial}{\partial \sigma}(\boldsymbol{r}). \tag{12}$$

Again, a word $w \in \mathcal{L}(\boldsymbol{r})$ if and only if $\varepsilon\left(\frac{\partial}{\partial w}(\boldsymbol{r})\right) = \varepsilon$. The set $\partial^+(\boldsymbol{r})$ satisfies the following inductive definition, for which a proof can also be found in [4].

Proposition 15 ([8, 14])

$$\begin{array}{ll} \partial^+(\emptyset) = \emptyset, & \partial^+(\boldsymbol{r}+\boldsymbol{s}) = \partial^+(\boldsymbol{r}) \cup \partial^+(\boldsymbol{s}), \\ \partial^+(\varepsilon) = \emptyset, & \partial^+(\boldsymbol{r}\boldsymbol{s}) = \partial^+(\boldsymbol{r})\boldsymbol{s} \cup \partial^+(\boldsymbol{s}), \\ \partial^+(\sigma) = \{\varepsilon\}, & \partial^+(\boldsymbol{r}^\star) = \partial^+(\boldsymbol{r})\boldsymbol{r}^\star. \end{array} \tag{13}$$

From this proposition, it easily follows that

Corollary 16 ([2, 14]) $|\partial^+(\boldsymbol{r})| \leq |\boldsymbol{r}|_\Sigma$.

Moreover, the size of each partial derivative is at most quadratic in the size of the expression.

Proposition 17 (Antimirov [2, Th. 3.8]) *For $\boldsymbol{r}' \in \partial^+(\boldsymbol{r})$, the size $\|\boldsymbol{r}'\|$ is $O((|\boldsymbol{r}|_\Sigma + 1)\|\boldsymbol{r}\|)$.*

The *partial derivative automaton* of $\boldsymbol{r} \in RE$ is defined as

$$\boldsymbol{a}_{\mathrm{PD}}(\boldsymbol{r}) = \langle \mathrm{PD}(\boldsymbol{r}), \Sigma, \delta_{\mathrm{PD}}, \boldsymbol{r}, F_{\mathrm{PD}} \rangle,$$

where $\mathrm{PD}(\boldsymbol{r}) = \partial^+(\boldsymbol{r}) \cup \{\boldsymbol{r}\}$, $F_{\mathrm{PD}} = \{\, \boldsymbol{r}' \in \mathrm{PD}(\boldsymbol{r}) \mid \varepsilon(\boldsymbol{r}') = \varepsilon \,\}$, and $\delta_{\mathrm{PD}}(\boldsymbol{r}', \sigma) = \frac{\partial}{\partial \sigma}(\boldsymbol{r}')$, for all $\boldsymbol{r}' \in \mathrm{PD}(\boldsymbol{r})$ and $\sigma \in \Sigma$. We have, $|\mathrm{PD}(\boldsymbol{r})| \leq |\boldsymbol{r}|_\Sigma + 1$.

Proposition 18 ([2, 14]) $\mathcal{L}(\boldsymbol{r}) = \mathcal{L}(\boldsymbol{a}_{\mathrm{PD}}(\boldsymbol{r}))$.

Broda et al. [3] showed that although the Brzozowski automaton is not obtained by the determinization of the partial derivative automaton, both automata can be related considering a quotient by a left-invariant relation on the sets of states.

One can adapt Algorithm 1 to use partial derivatives and obtain a standard regular expression for the quotient. We start, in line 2, with $S = \{(\{\boldsymbol{t}\}, \{\boldsymbol{r}\})\}$; in line 7 one needs to add to $\boldsymbol{s}$ all the expressions in the corresponding set; and in line 11 we compute the partial derivatives for all elements of the sets. We note that a finite set of expressions is equivalent to the sum of their elements. For completeness, the resulting algorithm is presented as Algorithm 2. The correctness of the algorithm follows from Lemma 12 adapted to partial derivatives. From Proposition 17, we have the following result.

Algorithm 2 Quotient of two regular expressions. Returns a regular expression.

```
 1: function QuotientPartial(t ,r)
 2:     S ← {({t}, {r})}
 3:     H ← ∅
 4:     s ← ∅
 5:     while (T, R) ←POP(S) do
 6:         if ε(T) = ε then
 7:             for r ∈ R do
 8:                 s ← s + r
 9:         PUSH(H, (T, R))
10:         for σ ∈ Σ do
11:             (T′, R′) ← (∂/∂σ (T) , ∂/∂σ (R))
12:             if (T′, R′) ∉ H then
13:                 PUSH(S,(T′, R′))
      return s
```

Theorem 19. *Given two regular expressions $\boldsymbol{r}, \boldsymbol{t} \in RE$, the regular expression outputted by Algorithm 2 represents $\mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$ and has size $O((\|\boldsymbol{r}\|^3)$.*

To directly compute a partial derivative automaton for the quotient of two given expressions $\boldsymbol{t}$ and $\boldsymbol{r}$, one can consider the regular expression that represents that quotient and obtain its set of partial derivatives from the sets $\partial^+(\boldsymbol{r}) \cup \{\boldsymbol{r}\}$ and $\partial^+(\boldsymbol{t}) \cup \{\boldsymbol{t}\}$. This is the goal of the next subsection.

4.3. *Partial Derivatives for Regular Expressions with a Quotient Operator*

Consider the set of regular expressions defined by the grammar (1) with the additional operator '$^{-1}$':

$$\boldsymbol{r} := \cdots \mid (\boldsymbol{r}^{-1}\boldsymbol{r}),$$

where $\mathcal{L}(\boldsymbol{t}^{-1}\boldsymbol{r}) = \mathcal{L}(\boldsymbol{t})^{-1}\mathcal{L}(\boldsymbol{r})$. We denote by $RE(^{-1})$ the resulting set of expressions.

By Lemma 1, Lemma 12, and Equation (11) we have

Corollary 20. *$\varepsilon(\boldsymbol{t}^{-1}\boldsymbol{r}) = \varepsilon$ if and only if there exists $w \in \Sigma^\star$ such that $\varepsilon(\frac{\partial}{\partial w}(\boldsymbol{t})) = \varepsilon(\frac{\partial}{\partial w}(\boldsymbol{r})) = \varepsilon$.*

We aim to define the set of partial derivatives by a word of an expression of the form $\boldsymbol{t}^{-1}\boldsymbol{r}$, assuming partial derivatives are defined for their subexpressions. From Lemma 12, to define $\frac{\partial}{\partial \sigma}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right)$, one needs to have the sets of partial derivatives of $\boldsymbol{t}$ and $\boldsymbol{r}$ by the same word.

First, let the set of *complete partial derivatives* of a regular expression $\boldsymbol{r}$ by a symbol $\sigma \in \Sigma$, $\frac{\partial^c}{\partial \sigma}(\boldsymbol{r})$, be as defined in Eq. (10) except that when the set is empty we add the regular expression $\emptyset$, i.e., $\frac{\partial^c}{\partial \sigma}(\emptyset) = \frac{\partial}{\partial \sigma}(\varepsilon) = \{\emptyset\}$ and $\frac{\partial}{\partial \sigma}(\sigma') = \{\emptyset\}$, if $\sigma \neq \sigma'$. The set of complete partial derivatives by a word $w \in \Sigma^\star$ is inductively defined by $\frac{\partial^c}{\partial \varepsilon}(\boldsymbol{r}) = \{\boldsymbol{r}\}$ and $\frac{\partial^c}{\partial w\sigma}(\boldsymbol{r}) = \frac{\partial^c}{\partial \sigma}\left(\frac{\partial^c}{\partial w}(\boldsymbol{r})\right)$.

Lemma 21. *Let $\boldsymbol{r} \in RE$, $\sigma \in \Sigma$, and $w \in \Sigma^\star$.*

(1) $\frac{\partial^c}{\partial \sigma}(\boldsymbol{r}) = \frac{\partial}{\partial \sigma}(\boldsymbol{r})$ *or* $\frac{\partial^c}{\partial \sigma}(\boldsymbol{r}) = \frac{\partial}{\partial \sigma}(\boldsymbol{r}) \cup \{\emptyset\}$.
(2) $\frac{\partial^c}{\partial w}(\boldsymbol{r}) = \frac{\partial}{\partial w}(\boldsymbol{r})$ *or* $\frac{\partial^c}{\partial w}(\boldsymbol{r}) = \frac{\partial}{\partial w}(\boldsymbol{r}) \cup \{\emptyset\}$.

Proof. The proof of (1) is by induction on the structure of $\boldsymbol{r}$. The proof of (2) is by induction on $|w|$. For ε it is obvious. Suppose the result holds for w and consider $w\sigma$. We have

$$\frac{\partial^c}{\partial w\sigma}(\boldsymbol{r}) = \frac{\partial^c}{\partial \sigma}\left(\frac{\partial^c}{\partial w}(\boldsymbol{r})\right).$$

By the inductive hypothesis, suppose $\frac{\partial^c}{\partial w}(\boldsymbol{r}) = \frac{\partial}{\partial w}(\boldsymbol{r}) \cup \{\emptyset\}$. Then

$$\begin{aligned}\frac{\partial^c}{\partial w\sigma}(\boldsymbol{r}) &= \frac{\partial^c}{\partial \sigma}\left(\frac{\partial^c}{\partial w}(\boldsymbol{r})\right) = \frac{\partial^c}{\partial \sigma}\left(\frac{\partial}{\partial w}(\boldsymbol{r}) \cup \{\emptyset\}\right) = \frac{\partial^c}{\partial \sigma}\left(\frac{\partial}{\partial w}(\boldsymbol{r})\right) \cup \{\emptyset\} \\ &= \bigcup_{\boldsymbol{r}' \in \frac{\partial}{\partial w}(\boldsymbol{r})} \frac{\partial^c}{\partial \sigma}(\boldsymbol{r}') \cup \{\emptyset\} = \bigcup_{\boldsymbol{r}' \in \frac{\partial}{\partial w}(\boldsymbol{r})} \frac{\partial}{\partial \sigma}(\boldsymbol{r}') \cup \{\emptyset\} = \frac{\partial}{\partial w\sigma}(\boldsymbol{r}) \cup \{\emptyset\}.\end{aligned}$$

If $\frac{\partial^c}{\partial w}(\boldsymbol{r}) = \frac{\partial}{\partial w}(\boldsymbol{r})$, we either have $\frac{\partial^c}{\partial w\sigma}(\boldsymbol{r}) = \frac{\partial}{\partial w\sigma}(\boldsymbol{r})$ or if there is $\boldsymbol{r}' \in \frac{\partial}{\partial w}(\boldsymbol{r})$ such that $\frac{\partial^c}{\partial \sigma}(\boldsymbol{r}') = \frac{\partial}{\partial \sigma}(\boldsymbol{r}') \cup \{\emptyset\}$ then $\frac{\partial^c}{\partial w\sigma}(\boldsymbol{r}) = \frac{\partial}{\partial w\sigma}(\boldsymbol{r}) \cup \{\emptyset\}$. □

It is obvious that $\mathcal{L}\left(\frac{\partial^c}{\partial w}(\boldsymbol{r})\right) = \mathcal{L}\left(\frac{\partial}{\partial w}(\boldsymbol{r})\right)$ and that the set of complete partial derivatives of a regular expression $\boldsymbol{r}$ by all words is finite.

Let $W^{\boldsymbol{r}\times\boldsymbol{t}} = \{w_1, \ldots, w_n\}$ be the set of lexicographically minimal words required to compute all pairs of complete partial derivatives of $\boldsymbol{r}$ and $\boldsymbol{t}$ by a same word. The set $W^{\boldsymbol{r}\times\boldsymbol{t}}$ can be computed by an algorithm similar to Algorithm 2 omitting lines 6, 7, and 8, and storing the minimal words used in each case. Given two regular expressions $\boldsymbol{t}$ and $\boldsymbol{r}$ we define[b]

$$\Pi(\boldsymbol{t},\boldsymbol{r}) = \bigcup_{w\in W^{\boldsymbol{r}\times\boldsymbol{t}}} \frac{\partial^c}{\partial w}(\boldsymbol{t}) \times \frac{\partial^c}{\partial w}(\boldsymbol{r}).$$

The set of *partial derivatives* of $\boldsymbol{t}^{-1}\boldsymbol{r}$ by $\sigma \in \Sigma$ is defined as follows:

$$\frac{\partial}{\partial \sigma}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right) = \bigcup_{(\boldsymbol{t}',\boldsymbol{r}') \in \Pi(\boldsymbol{t},\boldsymbol{r})} \varepsilon(\boldsymbol{t}') \frac{\partial}{\partial \sigma}(\boldsymbol{r}'), \tag{14}$$

which can be generalized for $w \in \Sigma^\star$ by

$$\frac{\partial}{\partial w}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right) = \bigcup_{(\boldsymbol{t}',\boldsymbol{r}') \in \Pi(\boldsymbol{t},\boldsymbol{r})} \varepsilon(\boldsymbol{t}') \frac{\partial}{\partial w}(\boldsymbol{r}'). \tag{15}$$

Note, that considering Corollary 20, one has $\varepsilon(\boldsymbol{t}^{-1}\boldsymbol{r}) = \varepsilon$ if and only if there exists $(\boldsymbol{t}',\boldsymbol{r}') \in \Pi(\boldsymbol{t},\boldsymbol{r})$ such that $\varepsilon(\boldsymbol{t}') = \varepsilon(\boldsymbol{r}') = \varepsilon$.

Proposition 22. *For $w \in \Sigma^\star$,*

$$\mathcal{L}\left(\frac{\partial}{\partial w}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right)\right) = w^{-1}\mathcal{L}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right).$$

Proof. We suppose that partial derivatives are defined for the subexpressions. Suppose that

$$v \in \mathcal{L}\left(\frac{\partial}{\partial w}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right)\right) = \bigcup_{(\boldsymbol{t}',\boldsymbol{r}') \in \Pi(\boldsymbol{t},\boldsymbol{r})} \mathcal{L}\left(\varepsilon(\boldsymbol{t}')\frac{\partial}{\partial w}(\boldsymbol{r}')\right),$$

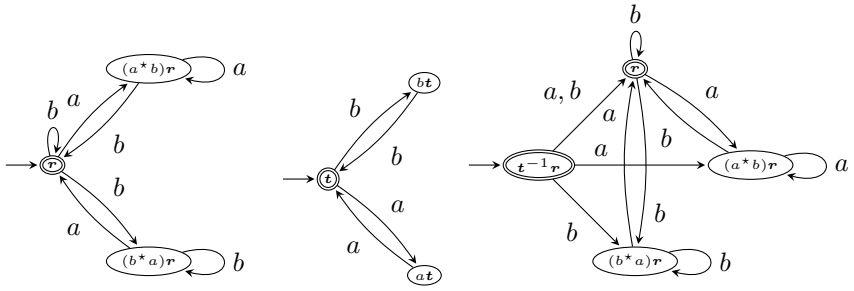
then there exist $(\boldsymbol{t}',\boldsymbol{r}') \in \Pi(\boldsymbol{t},\boldsymbol{r})$ such that $v \in \mathcal{L}\left(\frac{\partial}{\partial w}(\boldsymbol{r}')\right)$ and $\varepsilon(\boldsymbol{t}') = \varepsilon$. Then, there exists $u \in W^{\boldsymbol{r}\times\boldsymbol{t}}$ such that $\boldsymbol{t}' \in \frac{\partial}{\partial u}(\boldsymbol{t})$, $\boldsymbol{r}' \in \frac{\partial}{\partial u}(\boldsymbol{r})$, $u \in \mathcal{L}(\boldsymbol{t})$, and $uwv \in \mathcal{L}(\boldsymbol{r})$. Thus, we conclude that $v \in w^{-1}\mathcal{L}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right)$. Conversely, if $v \in w^{-1}\mathcal{L}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right)$ then $wv \in \mathcal{L}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right)$. Then, there exists $u \in W^{\boldsymbol{r}\times\boldsymbol{t}}$, such that there exist $\boldsymbol{t}' \in \frac{\partial}{\partial u}(\boldsymbol{t})$ and $\boldsymbol{r}' \in \frac{\partial}{\partial u}(\boldsymbol{r})$ such that $\varepsilon(\boldsymbol{t}') = \varepsilon$ and $v \in \frac{\partial}{\partial w}(\boldsymbol{r}')$. Thus, $v \in \mathcal{L}\left(\frac{\partial}{\partial w}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right)\right)$. □

Finally, we can conclude that the set of partial derivatives by non-empty words of $\boldsymbol{t}^{-1}\boldsymbol{r}$ is included in the set of partial derivatives of $\boldsymbol{r}$.

Proposition 23. $\partial^+(\boldsymbol{t}^{-1}\boldsymbol{r}) \subseteq \partial^+(\boldsymbol{r}) \cup \{\boldsymbol{r}\}$.

[b]The set $\Pi(\boldsymbol{t},\boldsymbol{r})$ can be computed from the set H in Algorithm 2. We note that those pairs containing $\emptyset$ are not needed for the case here studied as they correspond to an empty set of partial derivatives.

Fig. 3. From left to right, we have the NFAs $\boldsymbol{a}_{\mathrm{PD}}(\boldsymbol{r})$, $\boldsymbol{a}_{\mathrm{PD}}(\boldsymbol{t})$ and $\boldsymbol{a}_{\mathrm{PD}}(\boldsymbol{t}^{-1}\boldsymbol{r})$, respectively, as defined in Example 24.



The partial derivative automaton $\boldsymbol{a}_{\mathrm{PD}}(\boldsymbol{r})$ for expressions in $\boldsymbol{r} \in RE(^{-1})$ is defined as before. We note that Antimirov [2] already observed that $|\partial^+(\boldsymbol{t}^{-1}\boldsymbol{r})| \leq |\partial^+(\boldsymbol{r})| + 1$ but did not give a specific algorithm for the computation of those partial derivatives.

Example 24. *Let* $\boldsymbol{r} = (a^\star b + bb^\star a)^\star$ *and* $\boldsymbol{t} = (aa + bb)^\star$ *be as in Example 14. We have* $\partial^+(\boldsymbol{r}) = \{\boldsymbol{r}, (a^\star b)\boldsymbol{r}, (b^\star a)\boldsymbol{r}\}$ *and* $\partial^+(\boldsymbol{t}) = \{\boldsymbol{t}, a\boldsymbol{t}, b\boldsymbol{t}\}$. *Omitting the pairs with* $\emptyset$, *we have*

$$\Pi(\boldsymbol{t}, \boldsymbol{r}) = \{(\boldsymbol{t}, \boldsymbol{r}), (a\boldsymbol{t}, (a^\star b)\boldsymbol{r}), (b\boldsymbol{t}, \boldsymbol{r}), (b\boldsymbol{t}, (b^\star a)\boldsymbol{r}), (\boldsymbol{t}, (a^\star b)\boldsymbol{r}), (\boldsymbol{t}, (b^\star a)\boldsymbol{r}), (a\boldsymbol{t}, \boldsymbol{r})\}.$$

Then $\frac{\partial}{\partial a}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right) = \{\boldsymbol{r}, (a^\star b)\boldsymbol{r}\}$, $\frac{\partial}{\partial b}\left(\boldsymbol{t}^{-1}\boldsymbol{r}\right) = \{\boldsymbol{r}, (b^\star a)\boldsymbol{r}\}$, *and the remaining partial derivatives of* $\boldsymbol{t}^{-1}\boldsymbol{r}$ *coincide with the ones of* $\boldsymbol{r}$. *We depicted the respective partial derivative automata in Fig. 3.*

5. Experimental Results

The three algorithms presented in this paper for the quotient operation of two regular languages were implemented in Python within the FAdo package [17]. If the two regular languages are represented by NFAs $\boldsymbol{a}$ and $\boldsymbol{b}$, the algorithm described in Theorem 8 computes the NFA for the quotient of $\boldsymbol{a}$ w.r.t $\boldsymbol{b}$. We denote the resulting NFA by $\boldsymbol{a}_\times$. For regular expressions $\boldsymbol{r}, \boldsymbol{t} \in RE$, Algorithm 2 was implemented using partial derivatives. In the same way, we implemented the algorithm for the computation of a partial derivative automaton for $\boldsymbol{t}^{-1}\boldsymbol{r}$, presented in the previous section[c], and which we denote by $\boldsymbol{a}^{-1}_{\mathrm{PD}}$. For the manipulation of regular expressions and their partial derivatives, we used the PDDAG algorithm that not only builds a compressed representation of a regular expression (identifying common subexpressions) but also of its partial derivatives [13]. To test the feasibility of the derivative based methods, we run some preliminary experiments. In the experiments, we uniformly

[c]Or, in general, for expressions in $RE(^{-1})$.

and randomly generated expressions $\boldsymbol{r} \in RE$, in prefix notation and in *strong star normal form* (ssnf). A regular expression in ssnf does not have a subexpression $\boldsymbol{r}'$ with $\varepsilon(\boldsymbol{r}') = \varepsilon$ under a $\star$ operator. For this class of regular expressions, it was shown that on average the size of a partial derivative is linear on the size of the regular expression [12] (which contrasts with the quadratic worst case). For each expression size $n \in \{100, 200, 300, 500, 1000, 2000\}$, and alphabet size $k \in \{2, 10\}$, samples of 10000 expressions were generated. For each generated expression $\boldsymbol{r}$, we randomly choose an expression $\boldsymbol{t}$ from the same sample, and we run the three algorithms on the pair $(\boldsymbol{r}, \boldsymbol{t})$. The algorithms make use of the compressed representation of regular expressions and their derivatives (DAGs). For the first one, $\boldsymbol{a}_\times$, we started by computing the $\boldsymbol{a}_{\mathrm{PD}}$ for each expression. Some results are presented in Table 1. The upper part of the table has some values for expressions of alphabet size $k = 2$ and the lower part for $k = 10$. For each size n, we give the average value (first row) and the maximum value (second row) of each value computed. Each column (from the leftmost) contains the following values: n, the size of the generated expressions; $|PD(\boldsymbol{r})|$ the number the partial derivatives of $\boldsymbol{r}$; $|\boldsymbol{a}_\times|$ number of states of the NFA for $\boldsymbol{t}^{-1}\boldsymbol{r}$; $|\boldsymbol{a}_{\mathrm{PD}}^{-1}|$ number of states of $\boldsymbol{a}_{\mathrm{PD}}(\boldsymbol{t}^{-1}\boldsymbol{r})$; $\|\boldsymbol{s}\|$ the size of the (standard) regular expression outputted by Algorithm 1 and the last three columns have the time, in seconds, of the execution of each algorithm by pair of expressions. In the case of $k = 2$, we only have partial results as computations did not terminate in a reasonable time (due to some instances where the size of the resulting expression may be cubic w.r.t the size of $\boldsymbol{r}$). We can observe that on average, the times of execution of the three algorithms are similar. The size of the regular expression obtained by Algorithm 1 seems to be on average smaller than n^2. Moreover, it corresponds to a set of nodes of the DAG of $\boldsymbol{r}$, so not much extra-space is used.

6. Conclusions

We have revisited the concept of quotient between two languages from the perspectives of state complexity and of algorithmic construction of equivalent regular expressions and automata, where we have implemented and tested the presented algorithms.

We considered regular expressions with the quotient operation and extended the partial derivative automaton construction for those expressions. Here, we only did experiments for regular expressions where the quotient operation appears at the top level but we plan to extend them to general expressions in $RE(^{-1})$. The descriptional and computational complexity of these models deserve further study.

Many algorithms on automata have exponential worst-case performance because they need to determinise an NFA produced by the algorithm. How frequent are streams of DFAs that have very small NFAs? In this spirit, the concept of hard deterministic regular stream discussed in this paper deserves further attention.

Table 1. Experimental Results

$k = 2$							
n	$\lvert \mathrm{PD}(\boldsymbol{r})\rvert$	$\lvert \boldsymbol{a}_\times \rvert$	$\lvert \boldsymbol{a}_{\mathrm{PD}}^{-1}\rvert$	$\lVert \boldsymbol{s}\rVert$	$t_\times$	$t_{\boldsymbol{a}_{\mathrm{PD}}}$	$t_{\boldsymbol{s}}$
100	26	25	24	1328	0.01471	0.00180	0.00319
	37	37	36	9611	0.11817	0.07669	0.07964
200	50	49	47	6329	0.21657	0.04261	0.04979
	66	66	66	43182	1.50493	4.78903	4.76711
300	74	72	70	16050	1.15066	0.80065	0.82972
	93	93	93	125828	7.13557	249.60908	266.73566
$k = 10$							
n	$\lvert \mathrm{PD}(\boldsymbol{r})\rvert$	$\lvert \boldsymbol{a}_\times \rvert$	$\lvert \boldsymbol{a}_{\mathrm{PD}}^{-1}\rvert$	$\lVert \boldsymbol{s}\rVert$	$t_\times$	$t_{\boldsymbol{a}_{\mathrm{PD}}}$	$t_{\boldsymbol{s}}$
100	26	17	15	125	0.00111	0.00068	0.00077
	37	36	36	3118	0.01944	0.01603	0.01880
200	51	34	29	335	0.00345	0.00197	0.00224
	68	68	68	11549	0.04715	0.01863	0.02840
300	75	50	43	566	0.00689	0.00389	0.00435
	95	94	94	22666	0.09012	0.03374	0.04823
500	125	84	73	1255	0.01662	0.00945	0.01050
	153	153	153	97748	0.33206	0.22579	0.34803
1000	248	168	146	2967	0.05978	0.03964	0.04244
	280	279	278	275503	2.48369	1.12715	1.55901
2000	494	336	293	6723	0.25192	0.21578	0.22750
	550	550	543	937964	9.52470	8.83304	8.92168

References

[1] M. Almeida, N. Moreira and R. Reis, Testing equivalence of regular languages, *J. Autom. Lang. Comb.* **15**(1/2) (2010) 7–25.

[2] V. M. Antimirov, Partial derivatives of regular expressions and finite automaton constructions, *Theoret. Comput. Sci.* **155**(2) (1996) 291–319.

[3] S. Broda, M. Holzer, E. Maia, N. Moreira and R. Reis, Mesh of automata, *Inform. and Comput.* **265** (April 2019) 94–111.

[4] S. Broda, A. M. N. Moreira and R. Reis, Automata for regular expressions with shuffle, *Inform. and Comput.* **259** (2018) 162–173.

[5] J. A. Brzozowski, Quotient complexity of regular languages, *J. Autom. Lang. Comb.* **15**(1/2) (2010) 71–89.

[6] J. A. Brzozowski, Towards a theory of complexity of regular languages, *J. Autom. Lang. Comb.* **23**(1-3) (2018) 67–101.

[7] J. Brzozowski, Derivatives of regular expressions, *J. ACM* (11) (1964) 481–494.

[8] J. M. Champarnaud and D. Ziadi, From Mirkin's prebases to Antimirov's word partial derivatives, *Fundam. Inform.* **45**(3) (2001) 195–205.

[9] J. H. Conway, *Regular Algebra and Finite Machines* (William Clowes & Sons, Great Britain, 1971).

[10] M. Crochemore and C. Hancart, Automata for matching patterns, *Handbook of Formal Languages, Volume 2. Linear Modeling: Background and Application*, eds. G. Rozenberg and A. Salomaa (Springer, 1997), pp. 399–462.

[11] H. Gruber and M. Holzer, Language operations with regular expressions of polynomial size, *Theor. Comput. Sci.* **410**(35) (2009) 3281–3289.

[12] S. Konstantinidis, A. Machiavelo, N. Moreira and R. Reis, On the size of partial derivatives and the word membership problem, *Acta Informatica* **58** (2021) 357–375.
[13] S. Konstantinidis, A. Machiavelo, N. Moreira and R. Reis, Partial derivative automaton by compressing regular expressions, *Proc. 23rd DCFS 2021*, eds. Y. Yan and S. Ko *LNCS* **13037**, (Springer, 2021), pp. 100–112.
[14] B. G. Mirkin, An algorithm for constructing a base in a language of regular expressions, *Eng. Cybernetics* **5** (1966) 51—57.
[15] N. Moreira and R. Reis, Manipulation of regular expressions using derivatives: An overview, *Proc. 26th CIAA*, eds. P. Caron and L. Mignot *LNCS* **13266**, (Springer, 2022), pp. 19–33.
[16] A. Nerode, Linear automaton transformations, *Procedings of the American Mathematical Society* (1958) 541–544.
[17] Project FAdo, Tools for the manipulation of formal languages https://pypi.org/project/FAdo/, (Access date:1/4/2024).
[18] S. Yu, Q. Zhuang and K. Salomaa, The state complexities of some basic operations on regular languages, *Theor. Comput. Sci.* **125**(2) (1994) 315–328.